Implementación de un criptosistema de clave pública basado en estrellas de isogenias

Autor: Rubén Arias Martínez

Director: Josep M. Miret Biosca

Titulación: Ingeniería Informática

Universidad de Lleida

Escola Politècnica Superior

Septiembre de 2007

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La criptografía (del griego kryptos, ”ocultar”, y grafos, ”escribir”) es el arte o ciencia de cifrar y descifrar información utilizando técnicas matemáticas que hagan posible el intercambio de mensajes de manera que solamente puedan ser leídos por las personas a quienes van dirigidos.

Cuando se habla de la criptografía como ciencia se debería hablar de criptología, que engloba tanto las técnicas de cifrado, la criptografía propiamente dicha, como sus técnicas complementarias: el criptoanálisis, que estudia los métodos para romper textos cifrados con objeto de recuperar la información original en ausencia de la clave.

La finalidad de la criptografía es, en primer lugar, garantizar el secreto en la comunicación entre dos entidades (personas, organizaciones, etc.) y, en segundo lugar, asegurar que la información que se envía es auténtica en un doble sentido: que el remitente sea realmente quien dice ser y que el contenido del mensaje enviado, habitualmente denominado criptograma, no haya sido modificado en su tránsito.

En la actualidad, la criptografía no sólo se utiliza para el intercambio de información de forma segura ocultando su contenido a posibles fisgones. Una de las ramas de la criptografía que más ha revolucionado el panorama actual de las tecnologías informáticas es el de la firma digital: tecnología que busca asociar al emisor de un mensaje con su contenido de forma que aquel no pueda posteriormente repudiarlo.

En criptografía, la información original que debe protegerse se denomina *texto en claro*. El *cifrado* es el proceso de convertir el texto plano en un galimatías ilegible, denominado *texto cifrado* o *criptograma*. Por lo general, la aplicación concreta del algoritmo de cifrado se basa en la existencia de una *clave*. Las dos técnicas más básicas de cifrado en la criptografía clásica son la *sustitución* y *la trasposición*. El *descifrado* es el proceso inverso que recupera el texto en claro a partir del criptograma y la clave. El conjunto de protocolos, algoritmos de cifrado, procesos de gestión de claves y actuaciones de los usuarios, en su globalidad es lo que constituyen un criptosistema, que es con lo que el usuario final trabaja e interactúa.

Existen dos tipos de cifrado: los algoritmos que utilizan una única clave tanto en el proceso de cifrado como en el de descifrado y los que utilizan una clave para cifrar mensajes y una clave distinta para descifrarlos. Los primeros se denominan *cifrados simétricos* o de *clave simétrica* y son la base de los algoritmos de cifrado clásico. Los segundos se denominan *cifrados asimétricos* o de *clave pública* y forman el núcleo de las técnicas de cifrado modernas.

A menudo los procesos de cifrado y descifrado se encuentran en la literatura como *encriptado* y *desencriptado*,

aunque ambos son neologismos -anglicismos de los términos ingleses encrypt y decrypt- todavía sin reconocimiento académico.

Durante siglos, el principal uso de la criptografía, se redujo mayoritariamente a su uso en tiempos de guerra. Pero hoy en día el uso generalizado de redes informáticas en el tratamiento y transmisión de la información, así como el aumento del número de usuarios de estos sistemas, han motivado la necesidad de mejorar la seguridad en las comunicaciones. Son muchas y variadas las situaciones en las que se requiere garantizar la privacidad, la integridad o la autenticación de la información transmitida. Tales necesidades se han satisfecho usando distintos protocolos criptográficos.

Uno de los principales inconvenientes de la criptografía de clave compartida es la distribución de claves, es decir, la necesidad que dos usuarios tienen de pactar previamente su clave secreta si desean cifrar los mensajes que se van a enviar.

En 1976, dos ingenieros de la Universidad de Stanford, Whitfield Diffie y Martin Hellman [3] comenzaron una gran revolución, al diseñar la criptografía de clave pública. La nueva idea consistía en algo aparentemente imposible: un criptosistema en donde hubiera dos claves, una para cifrar y otra para descifrar. En la actualidad, el método de clave pública más utilizado probablemente sea el RSA, desarrollado en 1977 por Adi Shamir, Ronald Rivest y Leonard Adleman [16]. Se basa en que no existe un algoritmo lo suficientemente eficiente para factorizar grandes números que sean producto de dos números primos, su desventaja es la velocidad.

La Criptografía con Curvas Elípticas (CCE) es una variante de la criptografía de clave pública basada en las matemáticas de las curvas elípticas. Sus autores argumentan que la CCE puede ser más rápida y usar claves más cortas que los métodos antiguos - como RSA - al tiempo que proporcionan un nivel de seguridad equivalente. La utilización de curvas elípticas en criptografía fue propuesta de forma independiente por Neal Koblitz [7] y Victor Miller [12] en 1985.

Otro de los problemas en los que se basa la seguridad de los criptosistemas de clave pública es el problema del logaritmo discreto. Para ello es necesario buscar conjuntos finitos con una estructura de grupo abeliano y determinar su orden. El problema del logaritmo discreto, aunque es computacionalmente difícil, puede ser resuelto en tiempo polinomial usando el algoritmo de Shor [19] para un ordenador cuántico. Así, la mayoría de criptosistemas de clave pública actuales se volverán inseguros cuando el tamaño del registro cuántico sea suficiente. Por lo que resulta necesario el desarrollo de criptosistemas que sean fuertes frente a ordenadores cuánticos.

En este proyecto se propone un problema matemático que, hipotéticamente, es fuerte frente a ordenadores cuánticos. Consiste en la búsqueda de una isogenia entre curvas elípticas sobre un cuerpo finito. Además también se presenta un criptosistema de clave pública propuesto por Rostovtsev y Stolbunov[17], así como su implementación utilizando MAGMA.

Capítulo 2

Curvas elípticas

2.1. Curvas elípticas

Por los símbolos $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{C}$, $\mathbb{F}_p$, $\mathcal{R}[x]$ denotaremos el anillo de enteros, el cuerpo de los números racionales y el de los complejos, el cuerpo finito de p elementos y el anillo de polinomios con coeficientes en el anillo $\mathcal{R}$ respectivamente.

Sea K un cuerpo con característica diferente de 2 y 3. El *plano proyectivo* $\mathbb{P}^2_K$ es el conjunto de ternas $(X,Y,Z) \in K^3 \setminus \{0,0,0\}$ módulo la relación de equivalencia

$$(X,Y,Z) = (uX,uY,uZ)$$

para cualquier $u \in K^*$ arbitraria.

El eje $Z = 0$ se llama *eje del infinito*, y sus puntos, *puntos del infinito*.

Una curva elíptica $E(K)$ es una curva no singular, del plano proyectivo $\mathbb{P}^2_K$ dado por

$$Y^2Z = X^3 + AXZ^2 + BZ^3.$$

Esta curva se cruza con el eje del infinito en el punto $O_E = (0,1,0)$ con multiplicidad 3. Para los demás puntos podemos asumir $Z = 1$ y $x = \frac{X}{Z}, y = \frac{Y}{Z}$. De este modo la ecuación en el plano afín se puede reescribir de la siguiente forma

$$y^2 = x^3 + Ax + B,$$

y se llama *ecuación reducida de Weierstraβ*.

El polinomio $y^2 - (x^3 + Ax + B)$, el cual da la ecuación anterior, genera un ideal máximal de $K[x,y]$. A partir de este ideal se define *el cuerpo de funciónes de la curva*:

$$K(E) = K[x,y] \setminus (y^2 - (x^3 + Ax + B)).$$

2.2. Discriminante y j-invariante

La importancia del discriminante radica en el hecho de que nos informa si una ecuación de Weierstraβ concreta define una curva elíptica y se define de la siguiente forma:

$$\Delta = -16(4A^3 + 27B^2).$$

El j-invariante se aplica a la hora de demostrar isomorfismos entre curva elípticas. Su expresión es:

$$j(E) = \frac{-1728(4A)^3}{\Delta}$$

Estos dos hechos se muestran de una manera más formal en los siguientes teoremas.

Teorema 1. *Dada una ecuación de Weierstraβ, esta es no-singular (es decir, define una curva elíptica) si ,y sólo si,* $\Delta \neq 0$.

Teorema 2. *Si dos curvas elípticas* $E_1/\mathbb{F}$ *y* $E_2/\mathbb{F}$ *son isomorfas sobre el cuerpo* $\mathbb{F}$*, entonces* $j(E_1) = j(E_2)$*. En* $\overline{\mathbb{F}}$ *el inverso también es cierto.*

2.3. Curvas elípticas sobre $\mathbb{C}$

En esta sección se van a introducir algunos conceptos sobre retículos de complejos y la función j.

2.3.1. Retículos de complejos

Definición 1. *Un retículo (lattice) L de* $\mathbb{C}$ *es un subgrupo discreto de* $\mathbb{C}$ *generado por dos vectores* ω_1 *y* ω_2 *de manera que* $Im\frac{\omega_1}{\omega_2} > 0$*. Se denota* $L(\omega_1, \omega_2) = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$.

Sea $L(\omega_1, \omega_2) = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$ un retículo de complejos. Si $\tau = \frac{\omega_1}{\omega_2}$, con $\mathrm{Im}(\tau) > 0$, entonces podemos escribir

$$L(\omega_1, \omega_2) = \omega_2 L(1, \tau).$$

Definición 2. *Dado un entero k y un retículo de complejos L, definimos la serie*

$$G_k(L) = \sum_{\ell \in L, \ell \neq 0} \frac{1}{\ell^k}.$$

Esta serie es absolutamente convergente para $k > 2$.

Entonces definimos $a(L) = 60G_4(L)$ y $b(L) = 140G_6(L)$.

Teorema 3. *Dado un retículo de complejos L, la curva* $E(\mathbb{C})$ *de ecuación:*

$$y^2 = 4x^3 - a(L)x - b(L),$$

es una curva elíptica para la cual podemos establecer un isomorfismo de grupos:

$$\frac{\mathbb{C}}{L} \cong E(\mathbb{C}).$$

2.3.2. La función j

Sea $L = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$ un retículo y $\tau = \dfrac{\omega_1}{\omega_2}$ com $\text{Im}(\tau) > 0$, entonces el j-invariante de la curva asociado a $L(1,\tau)$ se define:

$$j(\tau) = 1728\frac{a^3(L)}{a(L)^3 - 27b(L)^3}.$$

2.4. Curvas elípticas sobre cuerpos finitos

A partir de esta sección se consideran curvas elípticas definidas sobre un cuerpo finito $\mathbb{F}_p$ donde p es un número primo.

Sea la curva elíptica $E/\mathbb{F}_p$ de ecuación

$$Y^2Z = X^3 + AXZ^2 + BZ^3.$$

Consideremos la curva en la clausura algebraica $\bar{\mathbb{F}}_p$. Entonces la aplicación

$$\begin{array}{rccc} \pi: & E(\bar{\mathbb{F}}_p) & \to & E(\bar{\mathbb{F}}_p) \\ & (x,y) & \to & (x^p, y^p) \end{array}$$

se denomina *endomorfismo de Frobenius* de la curva $E/\mathbb{F}_p$.

El endomorfismo de Frobenius de $E/\mathbb{F}_p$ satisface la ecuación característica sobre $\mathbb{C}$:

$$\pi^2 - t\pi + p = 0,$$

donde $t = p + 1 - \#E(\mathbb{F}_p)$ es la *traza de Frobenius*. Si $t^2 < 4p$, es decir, $\mid t \mid < 2\sqrt{p}$, el discriminante D_π de la ecuación anterior es negativo y se define como:

$$D_\pi = t^2 - 4p.$$

2.5. Polinomios de n-división

Sea $E/\mathbb{F}_p$ una curva elíptica.

Definicion 3. *Dado un entero $n > 0$, se define el grupo de n-torsión de la curva E como el subgrupo de puntos de la curva $E(\bar{\mathbb{F}}_p)$ tales que n es un múltiplo de su orden, es decir:*

$$E[n](\mathbb{F}_p) = \{P \in E(\mathbb{F}_p) \mid nP = O_E\}$$

y más en general

$$E[n] = \left\{P \in E(\bar{\mathbb{F}}_p) \mid nP = O_E\right\}$$

Los polinomios de división nos permiten encontrar los puntos de orden n de una curva $E/\mathbb{F}_p$. Las raíces de los polinomios de división nos dan las abscisas de estos puntos. Llamaremos Ψ_n al n-ésimo polinomio de división de una curva elíptica y cumple que

$$\Psi_n(P) = 0 \ \forall P \in E[n], \ P \neq O_E$$

Si consideramos la curva $E/\mathbb{F}_p$ de ecuación $y^2 = x^3 + Ax + B$, podemos definir los polinomios de división de la siguiente forma recursiva:

$$\begin{aligned}
\Psi_0(x,y) &= 0 \\
\Psi_1(x,y) &= 1 \\
\Psi_2(x,y) &= 2y \\
\Psi_3(x,y) &= 3x^4 + 6Ax^2 + 12Bx - A^2 \\
\Psi_4(x,y) &= 4y(x^6 + 5Ax^4 + 20Bx^3 - 5A^2x^2 - 4ABx - 8B^2 - A^3) \\
\Psi_{2n+1}(x,y) &= \tfrac{\Psi_n}{\Psi_2}(\Psi_{n+2}\Psi_{n-1}^2 - \Psi_{n-2}\Psi_{n+1}^2), \ n \geq 2 \\
\Psi_{2n}(x,y) &= \Psi_{n+2}\Psi_n^3 - \Psi_{n+1}^3\Psi_{n-1}, \ n \geq 3
\end{aligned}$$

Observamos que para n impar, Ψ_n es un polinomio de una única variable x.

Estos polinomios tienen una propiedad muy importante que nos permite calcular directamente los múltiplos de un punto:

$$nP = \left(x - \frac{\Psi_{n-1}\Psi_{n+1}}{\Psi_n^2}, \frac{\Psi_{n+2}\Psi_{n-1}^2 - \Psi_{n-2}\Psi_{n+1}^2}{4y\Psi_n^3}\right).$$

A partir del polinomio de division $\Psi_n(x,y) \in \mathbb{F}_p[x,y]$, podemos obtener el polinomio $f_n(x) \in \mathbb{F}_p[x]$ con propiedades similares de la siguiente forma:

- Dado que los polinomios $\Psi_n(x,y)$ estan definidos sobre puntos de la curva, podemos sustituir y^2 por $x^3 + ax + b$, obteniendo un nuevo polinomio $\Psi'_n(x)$ que cuando n es par queda multiplicado por y.
- Definimos el polinomio $f_n(x)$ como:

$$f_n(x) = \begin{cases} \Psi'_n(x) & \text{si } n \text{ es impar,} \\ \dfrac{\Psi'_n(x)y}{\Psi_2(x,y)} & \text{si } n \text{ es par.} \end{cases}$$

El grado del polinomio $f_n(x)$ es $\frac{n^2-1}{2}$ si n es impar y $\frac{n^2-4}{2}$ si n es par.

Estos nuevos polinomios continuan caracterizando los puntos de n-torsión de la forma siguiente:

Teorema 4. *Si $P = (x,y) \in E(\mathbb{F}_p)$, resulta que:*

$$nP = O_E \iff f_n(x) = 0.$$

Es decir las raíces de este polinomio son exactamente las abscisas de los puntos de n-torsión.

2.6. Isogenias

Dadas dos curvas elípticas E y E' sobre $\mathbb{F}_p$, una *isogenia* entre E y E' es un morfismo:

$$\begin{array}{rcl} I : E(\mathbb{F}_p) & \longrightarrow & E'(\mathbb{F}_p) \\ (x,y) & \longmapsto & (X(x,y), Y(x,y)) \\ O_E & \longmapsto & O_{E'} \end{array}$$

donde X e Y son expresiones racionales en las coordenadas (x,y) y tal que el elemento neutro de E satisface $I(O_E) = O_{E'}$. Se dice entonces que E y E' son curvas isógenas si existe una isogenia I entre ambas tal que $I(E) \neq \{O\}$.

Se denomina núcleo de de la isogenia I a los puntos $P \in E(\mathbb{F}_p)$ tales que $I(P) = O_{E'}$.

La multiplicación por $m \in \mathbb{Z}$ en E, $[m] : E \longrightarrow E$ es una isogenia de E en si misma. La isogenia nula en E se define de manera que $[0](P) = O$.

Las isogenias I, como todo morfismo entre variedades algebraicas, o son constantes, i.e., $I(E) = O$, o son exhaustivas, es decir $I(E) = E'$.

Teorema 5. *Sea I una isogenia, entonces para todo $P,Q \in E(\mathbb{F}_p)$ se tiene que $I(P+Q) = I(P) + I(Q)$.*

Ademas si I es una isogenia de E a E', entonces existe una única isogenia $\hat{I}$, llamada *dual* de I, de E' a E tal que:

$$\hat{I} \circ I = \ell_E \text{ y } I \circ \hat{I} = \ell_{E'},$$

donde ℓ_E es la operación de multiplicar por $\ell \in \mathbb{Z}_{>0}$ en E, es decir :

$$\begin{array}{rcl} \ell_E : E & \longrightarrow & E \\ P & \longmapsto & \ell P \end{array}$$

y a ℓ se le denomina *grado de la isogenia*.

Dadas dos isogenias de curvas elípticas $\varphi : E_1 \to E_2$ y $\psi : E_2 \to E_3$, se define la composición de isogenias de forma habitual como:

$$\psi\varphi : E_1 \to E_3$$

donde $deg(\psi\varphi) = deg(\psi)deg(\varphi)$ y $\widehat{\psi\varphi} = \hat{\psi}\hat{\varphi}$.

La composición de isogenias cumple la propiedad asociativa.

A partir de la cardinalidad de una curva elíptica se puede determinar las curvas que le son isógenas sobre $\mathbb{F}_p$.

Teorema 6. *Dos curvas elípticas son isógenas sobre $\mathbb{F}_p$ si y sólo si tienen el mismo número de puntos.*

Teorema 7. *Sea $E(\mathbb{F}_p)$ una curva elíptica que tiene como discriminante de Frobenius D_π y $(\frac{D_\pi}{\ell})$ sea el símbolo de Kronecker para algunos grados primos ℓ de isogenias. Si:*

- $\left(\frac{D_\pi}{\ell}\right) = -1$, *no hay isogenias de grado ℓ;*
- $\left(\frac{D_\pi}{\ell}\right) = 1$, *habrá dos isogenias de grado ℓ;*
- $\left(\frac{D_\pi}{\ell}\right) = 0$, *existiran 1 o $\ell+1$ isogenias de grado ℓ.*

El símbolo de Kronecker $\left(\frac{D_\pi}{\ell}\right)$ coincide si ℓ es un primo impar con el símbolo de Legendre. Si $\ell = 2$ se define de la siguiente forma:

$$\left(\frac{D_\pi}{\ell}\right) = \begin{cases} 1 & si \quad a \equiv \pm 1 \mod 8 \\ 0 & si \quad a \equiv 0 \mod 2 \\ -1 & si \quad a \equiv \pm 3 \mod 8 \end{cases}$$

2.6.1. Polinomio modular

Para un entero positivo n se define

$$D_n^* = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} : a,b,c,d \in \mathbb{Z},\ ad - bc = n,\ gcd(a,b,c,d) = 1 \right\},$$

y

$$S_n^* = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in D_n^* : d > 0,\ 0 \le b < d \right\}.$$

Se define el *polinomio modular* de orden n [1] como:

$$\Phi_n(X, j) = \prod_{\alpha \in S_n^*} (X - j \circ \alpha)$$

donde para cada $\alpha = \begin{pmatrix} a & b \\ 0 & d \end{pmatrix}$, $j \circ \alpha(\tau) = j\left(\frac{a\tau+b}{d}\right)$.

Teorema 8. *Sea n un entero positivo, el polinomio modular $\Phi_n(X, j)$ cumple las siguientes propiedades:*

1. $\Phi_n(X,Y) \in \mathbb{Z}[X,Y]$.
2. $\Phi_n(X,Y) = \Phi_n(Y,X)$ *si* $n > 1$.
3. $\Phi_n(X,Y)$ *es irreducible como polinomio en* X.
4. *Si n no es un cuadrado perfecto, entonces* $\Phi_n(X,Y)$ *es un polinomio de grado* > 1 *donde el coeficiente dominante es* ± 1.
5. *Si n es un primo p, entonces* $\Phi_p(X,Y) \equiv (X^p - Y)(X - Y^p)$ *mod* $p\mathbb{Z}[X,Y]$.

Teorema 9. *Existe una isogenia de grado n de E a E′, si y sólo si* $\Phi_n(j(E), j(E')) = 0$.

En el caso que $n = \ell$, un primo, hay precisamente $\ell + 1$ subgrupos del grupo de puntos de ℓ-torsión $E[\ell]$ de una curva E. Cada subgrupo es el kernel de una isogenia de grado ℓ.

Equivalentemente, cada subgrupo corresponde a una curva isógena con un j-invariante que es un cero en el polinomio $\Phi_\ell(x, j)$.

Notar que mientras el grado del polinomio modular $\Phi_\ell(x, j)$ es $\ell + 1$ en cualquier variable, sus coeficientes enteros pueden llegar a ser muy grandes cuando ℓ aumenta. Así que para $\ell > 10^6$, el cálculo de un polinomio modular es prácticamente infactible.

Ejemplo de polinomio modular para $\ell = 3$:

$$\begin{aligned}\Phi_3(x,y) \quad = \quad & x^4 - x^3y^3 + 2232x^3y^2 - 1069956x^3y \\ & + 36864000x^3 + 2232x^2y^3 + 2587918086x^2y^2 \\ & + 8900222976000x^2y + 452984832000000x^2 \\ & - 1069956xy^3 + 8900222976000xy^2 - 770845966336000000xy \\ & + 1855425871872000000000x + y^4 + 36864000y^3 \\ & + 452984832000000y^2 + 1855425871872000000000y.\end{aligned}$$

2.6.2. Cálculo de isogenias

Para calcular una curva elíptica isógena se puede usar el siguiente algoritmo [15]. Este toma como origen una curva elíptica $E_{a,b}$ con invariante j, un grado de isogenia ℓ y una raíz del polinomio $\Phi_\ell(X, j)$ como entrada y como salida da una curva elíptica $E_{a',b'}$.

1. Sean a, b, coeficientes de la curva $y^2 = x^3 + ax + b$, j el j-invariante de la curva y $\tilde{j}$ obtenido de la factorización del polinomio ℓ-modular $\Phi_\ell(x, j)$.
2. Calcular $E_4 = -48a$ y $E_6 = 864b$ en $\mathbb{F}_p$.
3. Calcular $j' = \frac{jE_6}{E_4}$ en $\mathbb{F}_p$.

4. Calcular $\tilde{j}' = -\dfrac{j'\Phi_x(j,\tilde{j})}{\ell\Phi_y(j,\tilde{j})}$ en $\mathbb{F}_p$, donde Φ_x y Φ_y son las derivadas parciales de Φ_ℓ.

5. Calcular

$$\tilde{a} = -\frac{1}{48}\frac{(\tilde{j}')^2}{\tilde{j}(\tilde{j}-1728)} \qquad \tilde{b} = \frac{1}{864}\frac{(\tilde{j}')^3}{\tilde{j}^2(\tilde{j}-1728)}$$

La curva $y^2 = x^3 + \tilde{a}x + \tilde{b}$ será isógena a $y^2 = x^3 + ax + b$. Consideramos la curva isomorfa $y^2 = x^3 + \ell^4\tilde{a}x + \ell^6\tilde{b}$.

Para calcular el núcleo de la isogenia entre $E_{A,B}$ y $E_{A',B'}$, podemos usar el código propuesto por MAGMA. Nos da el polinomio

$$K(X) = X^d + a_{d-1}X^{d-1} + \ldots + a_1X + a_0 \ \in \ \mathbb{F}_p[X].$$

donde $d = \frac{\ell-1}{2}$. Las raíces de $K(X)$ dan todas las coordenadas x de los puntos del núcleo.

2.6.3. Determinación de una dirección en un ciclo de isogenias

Sea $E/\mathbb{F}_p$ una curva elíptica con discriminante de Frobenius D_π y sea ℓ un primo de Elkies, es decir $\left(\frac{D_\pi}{\ell}\right) = 1$. Consideramos entonces dos isogenias I_1 e I_2 desde E, es decir:

$$E_1(\mathbb{F}_p) \xleftarrow{I_1} E(\mathbb{F}_p) \xrightarrow{I_2} E_2(\mathbb{F}_p).$$

El grupo de torsión $E[\ell]$ está formado por $\ell+1$ subgrupos de orden ℓ. Dos de estos subgrupos son los núcleos de I_1 y I_2.

El método para determinar la dirección en un ciclo de isogenias se comenta en [2]. Este método usa el comportamiento del endomorfismo de Frobenius en un núcleo de una isogenia. Cuando $\left(\frac{D_\pi}{\ell}\right) = 1$, el polinomio característico de Frobenius, considerado sobre $\mathbb{Z}/\ell\mathbb{Z}$, se descompone en factores lineales. Sean $\pi_1, \pi_2 \in \mathbb{Z}/\ell\mathbb{Z}$, raíces del polinomio. A π_1 y π_2 se les llama *valores propios de Frobenius*. El comportamiento del endomorfismo de Frobenius en el núcleo de una isogenia de grado ℓ es igual a la multiplicación de un punto por un valor propio:

$$(x^p, y^p) = \pi_i(x,y) \ \in \ \mathbb{F}_p[x,y] \bmod (y^2 - x^3 - Ax - B, \ K_i(x))$$

donde $y^2 = x^3 + Ax + B$ es la ecuación de la curva y $K_i(x)$ es el polinomio $K_i(X) = X^d + a_{d-1}X^{d-1} + \ldots + a_1X + a_0 \in \mathbb{F}_p[X]$, cuyas raíces dan las coordenadas x del núcleo de la isogenia I_i.

Así, π_1 corresponde a una dirección del ciclo, y π_2 a la otra.

Para determinar que núcleo corresponde a cada valor propio comprobamos que las abscisas de (x^p, y^p) y $\pi_i(x,y)$ sean iguales módulo la ecuación $K_i(x)$. Si esto no fuera suficiente porque se iguala para los dos núcleos, tendriamos que comprobar la misma igualdad, pero en este caso para las ordenadas.

2.6.4. Class number

Sea E una curva elíptica sobre $\mathbb{F}_p$. El anillo de endomorfismos de E se identifica con un orden en el cuerpo cuadrático imaginario $K = \mathbb{Q}(\sqrt{t^2-4p})$, donde t es la traza del discriminante de Frobenius de E.

En este caso el class number de K , $h(D_\pi)$ nos da el número de clases de isomorfía de curvas sobre $\mathbb{F}_p$ que tiene discriminante de Frobenius $D_\pi = t^2 - 4p$, es decir, el número de curvas isógenas con E.

Corolario 1. *Si un discriminante D_π es un producto de diferentes números primos, entonces el class number puede no ser primo.*

2.7. Estrella de isogenias

Sea $U = \{E_i(\mathbb{F}_p)\}$ un conjunto de curvas elípticas con el mismo número de puntos, de modo que cada elemento de U está únicamente determinado por un j-invariante de una curva elíptica. Según lo visto en el apartado anterior podemos calcular $\#U$ como el class number (supondremos que U contiene todas las clases de isomorfia con un mismo cardinal). Podemos considerar para cada curva, el conjunto U, como el conjunto de ℓ-isogenias desde E de ciertos grados primos prefijados.

Para una curva elíptica con invariante j, el número de isogenias que tienen grado primo ℓ es igual al número de raíces del polinomio modular. El número exacto de isogenias se puede calcular usando el teorema 7.

De acuerdo a este teorema, si

$$\left(\frac{D_\pi}{\ell}\right) = 1,$$

las isogenias de grado ℓ de curvas elípticas de U forman un ciclo. Cambiar la dirección de un ciclo significa el intercambio a isogenias duales.

Esta demostrado que cuando $\#U$ es primo y $\left(\frac{D_\pi}{\ell}\right) = 1$, todos los elementos de U forman un único ciclo de ℓ-isogenias.

Sea $\ell' \neq \ell$ otro grado de isogenias primo con la propiedad $\left(\frac{D_\pi}{\ell'}\right) = 1$. En este caso, las isogenias de grado ℓ' también forman un ciclo sobre U. De este modo podemos poner ambos ciclos uno encima de otro. Lo mismo se puede hacer para otros grados de isogenias del mismo tipo.

Definición 4. *Un grafo, consistente de un número primo de curvas elípticas, conectadas por isogenias de grado ℓ que satisfacen $\left(\frac{D_\pi}{\ell}\right) = 1$, es una estrella de isogenias.*

En la figura se muestra un ejemplo de una estrella. Hay 7 curvas elípticas sobre $\mathbb{F}_{83}$. Los nodos representan los j-invariantes.

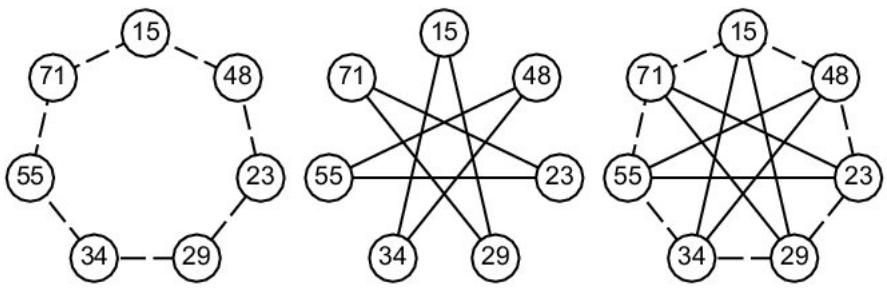

Figura 2.1: Ciclos de 3 y 5 isogenias y su estrella.

Si una estrella de isogenias es lo suficientemente grande, se puede usar para la construcción de criptosistemas, tal y como veremos en el siguiente capítulo. Para esto se necesita especificar una dirección en un ciclo.

2.7.1. Ruta en una estrella de isogenias

Sean S una estrella de isogenias, $L = \{\ell_i\}$ - un conjunto de primos de Elkies correspondientes a grados de isogenias y $F = \{\pi_i\}$ - un conjunto de valores propios de Frobenius, los cuales especifican una dirección para cada $\ell_i \in L$.

Definición 5. *Un conjunto $R = \{r_i\}$, donde r_i es el número de pasos por la isogenia de grado ℓ_i en la dirección π_i, es una ruta en la estrella de isogenias.*

Se pueden definir composiciones de rutas $A = \{a_i\}$ y $B = \{b_i\}$ como $AB = \{a_i + b_i\}$. Las rutas son conmutativas: $AB = BA$.

Capítulo 3

Criptosistema de clave pública basado en estrellas de isogenias

En este capítulo se muestran los algoritmos para cifrar y descifrar la información, así como los parámetros comunes al cifrado y al descifrado usando el criptosistema propuesto por Rostovtsev y Stolbunov [17]. También se comentan los criterios seguidos para el cálculo de esos parámetros comunes y en que basa su seguridad este criptosistema.

3.1. Algoritmos

3.1.1. Parámetros comunes

- $\mathbb{F}_p$, cuerpo finito de tamaño p sobre el que se trabajará;
- E_{init} - curva elíptica inicial, especificada por un par de coeficientes (A_{init}, B_{init}) de la ecuación $y^2 = x^3 + Ax + B$ sobre $\mathbb{F}_p$;
- d - número de grados de isogenias que seran usados;
- $L = \{\ell_i\},\ 1 \leq i \leq d$ - conjunto de grados de Elkies de isogenias que seran usados;
- $F = \{\pi_i\},\ 1 \leq i \leq d$ - conjunto de valores propios de Frobenius, que especifican una dirección para cada $\ell_i \in L$;
- k - límite de pasos de un grado de isogenia en una ruta. Para cada grado de isogenia, el número de pasos $\{r_i\}$ se selecciona por $0 \leq r_i \leq k$.

Clave Privada: Una ruta privada R_{priv}.

Clave Pública: Una curva elíptica calculada como $E_{pub} = R_{priv}(E_{init})$. Se especifica por (A_{pub}, B_{pub}).

3.1.2. Cifrado

Entrada:

- parámetros comunes del criptosistema: $\mathbb{F}_p,\ (A_{pub},\ B_{pub}),\ d,\ L,\ F,\ k$;
- E_{pub} - clave pública, especificada por (A_{pub}, B_{pub});
- $m \in \mathbb{F}_p$ - texto en claro.

Algoritmo:

1. Elegir una ruta R_{enc} aleatoria. Si $R_{enc} = \{0,0,...,0\}$, repetir este paso.
2. Calcular la curva de la estrella $E_{enc} = R_{enc}(E_{pub})$.
3. Calcular el texto cifrado $s = m * j_{enc}\ (mod\ p)$, donde j_{enc} es la j-invariante de E_{enc}.
4. Calcular la curva de la estrella $E_{add} = R_{enc}(E_{init})$.

Salida

- s - un texto cifrado;
- E_{add} - curva elíptica adicional especificada por (A_{add}, B_{add}).

3.1.3. Descifrado

Entrada

- parámetros comunes del criptosistema: $\mathbb{F}_p,\ (A_{pub},\ B_{pub}),\ d,\ L,\ F,\ k$;
- R_{priv} - clave privada;
- s - un texto cifrado;
- E_{add} - curva elíptica adicional especificada por (A_{add}, B_{add}).

Algoritmo

1. Calcular la curva de la estrella $E_{enc} = R_{priv}(E_{add})$.
2. Calcular el texto en claro $m = \frac{s}{j_{enc}}(mod\ p)$, donde j_{enc} es la j-invariante de E_{enc}.

Salida

- m - texto en claro.

3.2. Selección de parámetros del criptosistema

En esta sección se discute la selección de una curva inicial E_{init}, que determinará la estrella de isogenias.

Este algoritmo requiere el cálculo de la longitud de un ciclo de isogenias, lo que se convierte en el cálculo del class number para D_π.

De acuerdo al corolario 1, para obtener un class number primo se debe usar un discriminante primo.

En la práctica, un class number puede ser determinado usando métodos analíticos. En particular, se puede alcanzar una buena aproximación mediante la fórmula de [18]:

$$h(D_\pi) \approx \frac{\sqrt{|\, D_\pi \,|}}{3,14159} \prod_{p=2}^{P} \frac{p}{p - \frac{D_\pi}{p}}.$$

El producto se realiza sobre todos los números primos hasta algun primo grande P. Cuanto mayor sea P, mayor será la precisión de la estimación. Se puede obtener el valor exacto con una búsqueda de fuerza bruta cerca de la estimación.

El requisito de primalidad de $\#U$ se puede sustituir por el requisito de la existencia de un divisor primo muy grande. Entonces la complejidad del criptosistema se estima como $O(\sqrt{r})$ donde r es el divisor primo más grande de $\#U$.

De acuerdo al teorema 7, se debe cumplir que para cada valor ℓ_i, $\left(\frac{D_\pi}{\ell_i}\right) = 1$, y de este modo las ℓ_i-isogenias forman un ciclo.

Para minimizar la complejidad computacional del cifrado, se debería utilizar un número d de grados de isogenias igual a $O(log\ \#U)$. En este caso, el número máximo k de pasos por una isogenia no debería exceder de 2 (normalmente 1).

Para una curva elíptica $E(\mathbb{F}_p)$, la complejidad computacional de una isogenia de grado ℓ es $O(\ell(log\ p)^2)$ [2]. Por lo tanto, isogenias de grados pequeños son fáciles de calcular.

3.3. Seguridad del criptosistema

La seguridad de este criptosistema se basa en el problema de la búsqueda de una isogenia entre curvas elípticas. Para romper este criptosistema, se debería buscar una isogenia entre E_{init} y E_{pub} (o entre E_{init} y E_{add}). Para la búsqueda de esta isogenia se podrían utilizar las siguientes técnicas:

- Fuerza Bruta.
  Usando un grado de isogenia y moviendonos desde E_{init} hasta obtener E_{pub}.
  Otra técnica de este tipo consiste en la enumeración de todas las posibles rutas desde E_{init}, de acuerdo a las restricciones de L, d y k, hasta encontrar E_{pub}.
  La complejidad de estos ataques se estima como $O(n)$ cálculos de isogenias.

- Meet-in-the-middle.
  Sea n el tamaño de una estrella de isogenias. Cuando una estrella está formada por un único grado de isogenias, la longitud media de una ruta es $O(n)$. Cuando está formada por dos grados de isogenias, la

longitud es $O(\sqrt{n})$. Cuando esta formada por m grados de isogenias, la longitud de la ruta es $S_m \approx O(mn^{\frac{1}{m}})$. No es difícil darse cuenta que la función $S_m(m)$ tiene su mínimo $O(log\ \ n)$, cuando $m \approx O(log\ n)$.
Para este ataque se selecciona $m \approx O(log\ n)$ grados de isogenias, satisfaciendo el criterio de Elkies. En este caso, la longitud media de una ruta desde E_{init} hasta E_{pub} no excede S_m. Se construyen todas las rutas desde E_{init}, no más grandes que $\frac{S_m}{2}$, y se almacenan en una base de datos. Se seleccionan rutas al azar con la misma longitud del criterio y se aplican a E_{pub}, y se mira el resultado en la base de datos almacenada. Se podría tener éxito con una alta probabilidad, de acuerdo a la paradoja del cumpleaños. La complejidad de este ataque se estima como $O(\sqrt{n})$ cálculos de isogenias.

- El método descrito por Galbraith en [6], tiene complejidad $O(\sqrt[4]{p})$.

Una suposición sobre la dificultad de romper el criptosistema utilizando un ordenador cuántico se encuentra en la siguiente idea. Cada cálculo de una isogenia por lo menos incluye construir el polinomio modular $\Phi_\ell(x,y)$ y calcular una raíz.

Para calcular una cadena de q isogenias, se debería encontrar consecutivamente estos q polinomios. Debido a que el parámetro de la ecuación (j-invariante) cambia con cada paso, no se pueden paralelizar las operaciones para evitar q pasos.

Asi que, la seguridad del criptosistema de isogenias de curvas elípticas sobre $\mathbb{F}_p$ se estima como $O(\sqrt{n}) \approx O(\sqrt[4]{p})$, de manera que el coste es exponencial.

Capítulo 4

Implementación

En este capítulo se presenta el software y hardware utilizado para la implementación del criptosistema, así como la implementación del mismo.

4.1. Software y hardware utilizado

El software utilizado para la realización de este proyecto ha sido:

- Sistema operativo Ubuntu Feisty 7.04, versión del kernel: 2.6.20-16-generic.
- Editor de textos Gedit versión 2.18.1.
- Software matemático Magma versión 2.10-8 [10].
- Editor de LaTeX Texmaker versión 1.5.

Las pruebas se han realizado sobre un portátil Dell Inspiron 6400 con 2 Ghz de memoria RAM y un procesador Intel Core Duo T2400 a 1,83 Ghz.

4.2. MAGMA

Magma es un lenguaje de programación diseñado para la investigación de estructuras algebracias, geométricas y combinatorias, o *magmas*. La sintáxis del lenguaje es parecida a otros lenguajes de programación más conocidos. Lo que hace especial a Magma es la cantidad de tipos de datos matemáticos tales como grupos, anillos, cuerpos, conjuntos, secuencias, etc junto con una gran colección de funciones para mejorar las operaciones estandares en álgebra. La información sobre los magmas y sus elementos se almacena de una forma matemática muy poderosa, haciendo factible el cálculo algebraico avanzado. Magma es una sofisticada herramienta para la experimentación, educación y para pruebas automatizadas que es útil tanto para estudiantes como para expertos matemáticos.

4.3. El código paso a paso

En este apartado se comentan las partes más relevantes del código.

La implementación esta compuesta de cuatro ficheros:

- *setup.m*: Contiene el cálculo de los parámetros comunes del criptosistema, así como de la clave pública y la clave privada.
- *cifrado.m*: Se encarga de realizar el cifrado del mensaje en claro.
- *descifrado.m*: Realiza el descifrado del mensaje cifrado.
- *funciones.m*: Contiene el cálculo de la isogenia, su kernel, la dirección y algunas funciones más que son comunes a los tres ficheros anteriores.

Durante la ejecución del programa se generan o utilizan varios ficheros de textos:

- *parametros.txt*: Contiene los parámetros comunes. Lo genera setup.m.
- *ClavePublica.txt*: Contiene la clave pública. Lo genera setup.m.
- *ClavePrivada.txt*: Contiene la clave privada. Lo genera setup.m.
- *mensaje.txt*: Contiene el mensaje en claro.
- *men_cifrado.txt*: Contiene el mensaje cifrado. Lo genera cifrado.m.
- *men_descifrado.txt*: Contiene el mensaje descifrado, que debe coincidir con el que hay en mensaje.txt. Lo genera descifrado.m.

4.3.1. Curva elíptica inicial

Para calcular la curva elíptica inicial hay que tener en cuenta lo siguiente:

- El class number deber ser primo.
- El símbolo de Kronecker del discriminante D_π sobre ℓ_i debe ser 1. Con esto se consigue que solamente haya dos isogenias.

Los pasos a seguir son los siguientes:

1. Calcular el tamaño p del cuerpo finito $\mathbb{F}_p$, donde p debe ser primo.

```
CalculaP:= function()
           paux:=Random(10^20,10^(21) - 1);
           p:=NextPrime(paux);
           return p;
end function;
```

`Random(a,b)` nos devuelve un entero en el intervalo [a...b]. Una vez obtenido un entero, `NextPrime(n)` nos devuelve el siguiente primo.

2. Calcular la curva elíptica sobre el cuerpo creado.

```
CurvaInicial:= function(p)
               repeat
                      a:=Random(p);
                      b:=Random(p);
               until IsEllipticCurve([GF(p)|a,b]);
               e:=EllipticCurve([GF(p)|a,b]);
               return e;
end function;
```

`IsSingular(E)` devuelve *true* si la curva E es no singular (Discriminante distinto de cero).

3. Se calcula el discriminante de Frobenius de la curva como se ha explicado anteriormente.

```
Discriminante:= function(e,p)
                t:=TraceOfFrobenius(e);
                D:=t^2-4*p;
                return D;
end function;
```

4. Se calcula el Class Number del cuerpo cuadrático asociado a la curva.

```
GetClassNumber:= function(D)
                 k:=QuadraticField(D);
                 cn:=ClassNumber(k);
                 return cn;
end function;
```

Primero definimos un cuerpos cuadrático imaginario con discriminante D. Una vez definido, ya podemos calcular el class number con la función proporcionada por Magma.

5. Se repiten todos los pasos anteriores mientras el class number no sea primo y no se cumpla el símbolo de Kronecker para todos los valores de ℓ. Para que el class number sea primo el discriminante debe ser primo, de esta manera no calculamos el class number hasta encontrar una curva elíptica con discriminante primo.

```
l:=[3,5,7,11,13,17];
repeat
       repeat
              p:=CalculaP();
              e:=CurvaInicial(p);
              D:=Discriminante(e,p);
       until IsPrime(D) and SimKronecker(D,l);
       cn:=GetClassNumber(D);
until IsPrime(cn);
```

`SimKronecker(D,l)` es una de las funciones creadas para este trabajo que calcula el símbolo de Kronecker para todas las ℓ_i y devuelve true o false si se cumple o no el criterio de Elkies, para ello utilizamos la funcion de Magma `KroneckerSymbol(D,l)`.

4.3.2. Valores propios de Frobenius

Para calcular el vector $F = \{\pi_i\},\ 1 \leq i \leq d$, que es el conjunto de valores propios de Frobenius, que especifican una dirección para cada $\ell_i \in L$ se procede como sigue. Como $\left(\frac{D_\pi}{\ell}\right) = 1$, el polinomio característico de Frobenius tendrá dos raíces sobre $\mathbb{Z}/\ell\mathbb{Z}$, de las cuales se elige una de ellas.

```
ValoresPropios:= function(e,l,p)
                 vp:=[0];
                 for i in [1..#l] do
                          pr:=PolynomialRing(GF(l[i]));
                          t:=TraceOfFrobenius(e);
                          pol:=x^2-t*x+p;
                          sol:=Roots(pol);
                          vp[i]:=sol[1][1];
                 end for;
                 return vp;
end function;
```

4.3.3. Cálculo de una isogenia

Para calcular la isogenia se ha utilizado un código que ya viene implementado en Magma. Este código forma parte de la implementacion del algoritmo SEA. Se encuentra dentro de la carpeta *magma/package/Geometry/CrvEll/SEA/* en el fichero *elkies.m*.

A continuación se muestra una manera de calcular una isogenia de grado ℓ a partir de los coeficientes de una curva dada y de su j-invariante [15].

1. Sean a, b, coeficientes de la curva $y^2 = x^3 + ax + b$, j el j-invariante de la curva y $\tilde{j}$ obtenido de la factorización del polinomio ℓ-modular $\Phi_\ell(x, j)$.
2. Calcular $E_4 = -48a$ y $E_6 = 864b$ en $\mathbb{F}_p$.
3. Calcular $j' = \dfrac{jE_6}{E_4}$ en $\mathbb{F}_p$.
4. Calcular $\tilde{j}' = -\dfrac{j'\Phi_x(j,\tilde{j})}{\ell\Phi_y(j,\tilde{j})}$ en $\mathbb{F}_p$, donde Φ_x y Φ_y son las derivadas parciales de Φ_ℓ.
5. Calcular
$$\tilde{a} = -\frac{1}{48}\frac{(\tilde{j}')^2}{\tilde{j}(\tilde{j}-1728)} \qquad \tilde{b} = \frac{1}{864}\frac{(\tilde{j}')^3}{\tilde{j}^2(\tilde{j}-1728)}$$

La curva $y^2 = x^3 + \tilde{a}x + \tilde{b}$ será isógena a $y^2 = x^3 + ax + b$. Consideramos la curva isomorfa $y^2 = x^3 + \ell^4\tilde{a}x + \ell^6\tilde{b}$.

Para calcular el kernel de la isogenia, seguimos los siguientes pasos:

1. Calcular $\tilde{E}_4 = -48\tilde{a}$ y $\tilde{E}_6 = 864\tilde{b}$ en $\mathbb{F}_p$.
2. Calcular
$$m = -\frac{j'\Phi_{xx}(j,\tilde{j}) + 2\ell j'\tilde{j}'\Phi_{xy}(j,\tilde{j}) + \ell^2\tilde{j}'^2\Phi_{yy}(j,\tilde{j})}{j'\Phi_x(j,\tilde{j})}.$$
3. Calcular
$$p_1 = m\frac{\ell}{2} + \frac{\ell}{4}\left(\frac{E_4^2}{E_6} - \ell\frac{\tilde{E}_4^2}{\tilde{E}_6}\right) + \frac{\ell}{3}\left(\frac{E_6}{E_4} - \ell\frac{\tilde{E}_6}{\tilde{E}_4}\right)$$

   El coeficiente p_1 es el sumatorio de las abscisas de los puntos no triviales del kernel:
$$p_1 = S_1 = \sum_{P\in C - P_\infty} x(P).$$
4. Calcular $c_1 = -a/5$ y $c_2 = -b/7$.
5. Calcular
$$c_k = \frac{3}{(k-2)(2k+3)}\sum_{j=1}^{k-2} c_j c_{k-1-j} \qquad 3 \le k \le \frac{\ell-1}{2}.$$
6. Calcular $\tilde{c}_k$ con la misma fórmula, pero con $\ell^4\tilde{a}$ y $\ell^4\tilde{b}$.
7. Considerar
$$exp\left(-\frac{1}{2}p_1 z^2 - \sum_{k=1}^{\infty}\frac{\tilde{c}_k - \ell c_k}{(2k+1)(2k+2)}z^{2k+2}\right)$$
   y sea $A(w)$ sus series de potencias en $w = z^2$
8. Sea $C(w) = \sum_{k\ge 1} c_k w^k$.
   Ponemos $d = (\ell - i)/2$. El polinomio que se busca es $x^d + \sum_{i=0}^{d-1} F_i x^i$. Los coeficientes de este polinomio nos daran los simétricos de las abscisas del kernel. Inicializamos $F_d = 1$ y la recursividad dada por los coeficientes es
$$F_{d-i} = [A(w)]_i - \sum_{k=1}^{i}\left(\sum_{j=0}^{k}\binom{d-i+k}{k-j}[C(w)^{k-j}]_j\right)F_{d-i+k}.$$

donde $[\ ]_i$ denota el coeficiente de w_i en sus correspondientes series de potencias.

9. En particular

$$\begin{aligned} F_{d-1} &= -\frac{p_1}{2} \\ F_{d-2} &= \frac{p_1^2}{8} - \frac{\tilde{c}_1 - \ell c_1}{12} - \frac{\ell-1}{2}c_1 \\ F_{d-3} &= -\frac{p_1^3}{48} - \frac{\tilde{c}_2 - \ell c_2}{30} + p_1\frac{\tilde{c}_1 - \ell c_1}{24} - \frac{\ell-1}{2}c_2 + \frac{\ell-3}{4}c_1 p_1. \end{aligned}$$

4.3.4. Dirección de una isogenia

Para calcular la dirección de una isogenia se utiliza el comportamiento del endomorfismo de Frobenius en el núcleo de la isogenia. Como ya se ha explicado anteriormente, tenemos que:

$$(x^p, y^p) = \pi_i(x,y) \in \mathbb{F}_p[x,y] \ mod\ (y^2 - x^3 - Ax - B,\ K_i(x))$$

$$nP = \left(x - \frac{\Psi_{n-1}\Psi_{n+1}}{\Psi_n^2}, \frac{\Psi_{n+2}\Psi_{n-1}^2 - \Psi_{n-2}\Psi_{n+1}^2}{4y\Psi_n^3} \right).$$

$$f_n(x) = \begin{cases} \Psi_n'(x) & \textit{si n es impar,} \\ \dfrac{\Psi_n'(x)y}{\Psi_2(x,y)} & \textit{si n es par.} \end{cases}$$

donde $K_i(x)$ es el núcleo de una isogenia y π_i es un valor propio de Frobenius.

Por lo tanto se tiene que cumplir que:

$$\begin{aligned} x^p &= (x - \tfrac{\Psi_{\pi_i-1}\Psi_{\pi_i+1}}{\Psi_{\pi_i}^2}) && mod \quad K_i(x) \\ y^p &= (\tfrac{\Psi_{\pi_i+2}\Psi_{\pi_i-1}^2 - \Psi_{\pi_i-2}\Psi_{\pi_i+1}^2}{4y\Psi_{\pi_i}^3}) && mod \quad K_i(x) \end{aligned}$$

Utilizando $f_n(x)$ en lugar de $\Psi_n(x,y)$, donde $f_n(x)$ son los polinomios de división reducidos, las ecuaciones anteriores nos quedan:

$$x^p = \begin{cases} (x - \dfrac{f_{\pi_i-1}f_{\pi_i+1}}{4(x^3+ax+b)f_{\pi_i}^2}) & mod \quad K_i(x) \quad si \quad \pi_i \textit{ es par} \\ (x - \dfrac{4(x^3+ax+b)f_{\pi_i-1}f_{\pi_i+1}}{f_{\pi_i}^2}) & mod \quad K_i(x) \quad si \quad \pi_i \textit{ es impar} \end{cases}$$

$$y^p = \begin{cases} (\dfrac{f_{\pi_i+2}f_{\pi_i-1}^2 - f_{\pi_i-2}f_{\pi_i+1}^2}{16(x^3+ax+b)^2 f_{\pi_i}^3}) & mod \quad K_i(x) \quad si \quad \pi \textit{ es par.} \\ (\dfrac{f_{\pi_i+2}f_{\pi_i-1}^2 - f_{\pi_i-2}f_{\pi_i+1}^2}{f_{\pi_i}^3}) & mod \quad K_i(x) \quad si \quad \pi \textit{ es impar.} \end{cases}$$

Y pasando las expresiones correspondientes a la abscisa a un lado, nos queda:

$$\begin{cases} 4(x^3+ax+b)(x^p-x)f_{\pi_i}^2 + f_{\pi_i-1}f_{\pi_i+1} = 0 \quad mod \quad K_i(x) \quad si \quad \pi_i \text{ es par} \\ \\ (x^p-x)f_{\pi_i}^2 + 4(x^3+ax+b)f_{\pi_i-1}f_{\pi_i+1} = 0 \quad mod \quad K_i(x) \quad si \quad \pi_i \text{ es impar} \end{cases}$$

Y para la ordenada, teniendo en cuenta que $y^p = (x^3+ax+b)^{\frac{p-1}{2}}$, nos queda:

$$\begin{cases} 16(x^3+ax+b)^{\frac{p+3}{2}} f_{\pi_i}^3 - f_{\pi_i+2}f_{\pi_i-1}^2 + f_{\pi_i-2}f_{\pi_i+1}^2 = 0 \quad mod \quad K_i(x) \quad si \quad \pi_i \text{ es par.} \\ \\ (x^3+ax+b)^{\frac{p-1}{2}} f_{\pi_i}^3 - f_{\pi_i+2}f_{\pi_i-1}^2 + f_{\pi_i-2}f_{\pi_i+1}^2 = 0 \quad mod \quad K_i(x) \quad si \quad \pi_i \text{ es impar.} \end{cases}$$

Por último nos quedaremos con el kernel que cumpla estas dos igualdades para el valor π_i escogido. Para calcular la isogenia a partir del kernel se utiliza la función `IsogenyFromKernel(E,K(x))` de Magma, donde E es la curva elíptica y $K(x)$ es el kernel de la isogenia.

4.3.5. Ruta en una estrella de isogenias

Hay que comentar que no se obtiene toda la estrella de isogenias ya que es un cálculo muy costoso. Para obtener una curva elíptica de la estrella, simplemente calculamos el número de isogenias que marque la ruta para cada ciclo.

Por ejemplo, si nos movemos en el sentido horario en la estrella de la figura 4.1 y tenemos la ruta $R = \{4,2\}$ y $L = \{3,5\}$, empezaremos desde el nodo 15 en el ciclo de 3-isogenias y nos moveremos 4 posiciones, es decir, calcularemos 4 3-isogenias. Del 15 iremos al 48, del 48 al 23, del 23 al 29 y del 29 al 34. Por último desde el ciclo de 5-isogenias nos moveremos 2 posiciones. Empezando desde el 34 iremos al 15 y del 15 al 29, que sera el j-invariante de la curva que estamos buscando. En total habremos calculado un total de 6 isogenias.

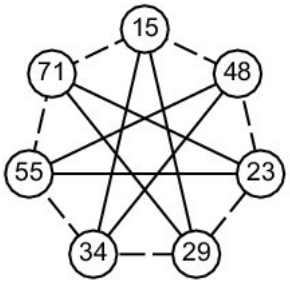

Figura 4.1: Estrella de isogenias de tamaño 2

Las líneas discontinuas forman el ciclo de 3-isogenias y las continuas el ciclo de 5-isogenias.

4.3.6. Número máximo de carácteres que se pueden cifrar

Si el mensaje original en base 10 es más grande que p, entonces deberemos particionar el mensaje original en trozos e ir cifrando cada bloque individualmente. Por lo tanto tendremos que calcular cual es el número máximo

de carácteres que pasados a enteros sean menor o igual a p. Sabemos que

$$\sum_{i=0}^{s-1}(int)caracter_i \cdot 256^i \leq n-1$$

donde s = número máximo de carácteres, y sabiendo que un carácter en código ASCII extendido tiene un valor máximo de 255, podemos tomar este valor y sustituirlo en la expresión anterior, que quedará de la siguiente forma

$$\sum_{i=0}^{s-1} 256^{i+1} = \sum_{i=1}^{s} 256^i = \frac{256-256^{s+1}}{1-256} \leq n-1,$$

y despejando s queda:

$$s \leq \frac{1}{8} \cdot log_2(255n+1) - 1.$$

4.3.7. De carácter a entero

Será necesario transformar el mensaje original, que estará formado por caracteres ASCII, a un número en base 10. Para ello tenemos la siguiente fórmula:

$$\begin{aligned} men-base-10 \quad &= \quad (int)caracter_0 * 256^0 + (int)caracter_1 * 256^1 + ... \\ &\quad\quad ... + (int)caracter_n * 256^n. \end{aligned}$$

4.3.8. De entero a carácter

Una vez obtenido m, el mensaje descifrado en base 10, tenemos que pasarlo a base 256. A continuación se muestra un ejemplo de como hacerlo:

$$\begin{array}{rrrcl} men\_base\_10 & div & 256 & \Rightarrow & cociente_0 \\ men\_base\_10 & mod & 256 & = & int \rightarrow caracter\ ASCII \\ cociente_0 & div & 256 & \Rightarrow & cociente_1 \\ cociente_0 & mod & 256 & = & int \rightarrow caracter\ ASCII \\ & & \vdots & & \end{array}$$

Capítulo 5

Resultados y conclusiones

En este capítulo se mostrará un ejemplo del funcionamiento del criptosistema. Además también se mostrarán los resultados temporales obtenidos y la relación entre el mensaje original y el mensaje cifrado, así como las conclusiones finales y las futuras líneas de trabajo.

5.1. Ejemplo

En esta sección se muestra un ejemplo del funcionamiento del criptosistema sobre una estrella de isogenias sobre un cuerpo $\mathbb{F}_p$ para un primo p de 10 dígitos.

Parametros comunes:

- $\mathbb{F}_{3750811091}$;
- $E_{init} = (178309677, 9207654)$, $j_{init} = 3466629689$, $D_\pi = -14794152764$, Class Number= 128053;
- $d = 6$ (número de grados de isogenias);
- $L = \{3,5,7,11,13,17\}$ (conjunto de grados de isogenias);
- $F = \{1,2,2,1,6,1\}$ (conjunto de valores propios que determinan las direcciónes);
- $0 \leqslant r_i \leqslant 10$;

Clave Privada $R_{priv} = \{1,8,0,10,6,1\}$.
Clave Pública $E_{pub} = (2555308664, 1350080485)$. (Ver tabla 5.1).

ℓ	paso	A	B
3	1	1568139640	430408011
5	1	3192577844	601468319
	2	3654760769	3273010665
	3	162366136	829413957
	4	1717477016	620070919
	5	2548411175	1093688189
	6	2383294126	2819651417
	7	623211615	801519606
	8	2783441106	83023120
11	1	313993079	3419588770
	2	715531863	1623109919
	3	3455465085	1800393262
	4	1133147708	564864171
	5	33036106	1179698423
	6	588948071	3556356435
	7	241398678	3721791122
	8	2405057786	2372567807
	9	2442245220	3304281885
	10	3706751144	908022641
13	1	3242842193	1642179036
	2	859678013	357656558
	3	2935627134	838348847
	4	1822317404	82165459
	5	2610875540	249042073
	6	275354662	1059730571
17	1	2555308664	1350080485

Tabla 5.1: Cálculo de E_{pub}.

Cifrado:

Ciframos el mensaje *EPS* que en base 10 es $m = 5460037$.

- $R_{enc} = \{1,9,2,0,1,1\}$
- $E_{enc} = (2162794285, 2052068526)$, $j_{enc} = 2761359959$. (Ver tabla 5.2).
- $s \equiv 5460037 \cdot 2761359959 \bmod 3750811091 = 3456399056$.
- $E_{add} = (1211018421, 2555314729)$. (Ver tabla 5.3).

Descifrado:

- $E_{enc} = (2162794285, 2052068526)$, $j_{enc} = 2761359959$. (Ver tabla 5.4).
- $m \equiv \dfrac{3456399056}{2761359959} \bmod 3750811091 = 5460037$.

ℓ	paso	A	B
3	1	2262815571	2724722478
5	1	1308470056	610116925
	2	555999245	2789689892
	3	3510806117	2537780138
	4	2416841277	3244313009
	5	3159357057	835630329
	6	2720443827	29552692
	7	2689756165	2300667872
	8	3524461124	130772141
	9	2823979963	414198124
7	1	2306953226	3165837855
	2	1637226286	2152950326
13	1	478060606	3008361191
17	1	2162794285	2052068526

Tabla 5.2: Cálculo de E_{enc}.

ℓ	paso	A	B
3	1	1568139640	430408011
5	1	3192577844	601468319
	2	3654760769	3273010665
	3	162366136	829413957
	4	1717477016	620070919
	5	2548411175	1093688189
	6	2383294126	2819651417
	7	623211615	801519606
	8	2783441106	83023120
	9	3501065130	1204328553
7	1	1702396066	895497953
	2	1526354376	1579394321
13	1	2222223722	137367232
17	1	1211018421	2555314729

Tabla 5.3: Cálculo de E_{add}.

ℓ	paso	A	B
3	1	3272433282	1131296566
5	1	233369431	104938799
	2	1873446358	1955874920
	3	2604117126	3036034355
	4	1046748863	3639317846
	5	1302080489	1445914661
	6	160740270	343794643
	7	667324903	667324903
	8	2938570182	2839999434
11	1	77009214	2551671546
	2	3397977526	499703241
	3	3456361827	847538760
	4	216446147	2053214388
	5	1707669396	486231743
	6	1865030584	699827741
	7	1574196395	2145874038
	8	1562719764	3599666009
	9	1693442976	1514397697
	10	1013692514	33948745
13	1	885705798	2850916651
	2	26561985	2580062230
	3	2100321435	2704134936
	4	1008870242	1801531565
	5	1637226286	2152950326
	6	478060606	3008361191
17	1	2162794285	2052068526

Tabla 5.4: Cálculo de E_{enc}.

5.2. Resultados

En este apartado se muestran una serie de resultados temporales. Las pruebas se realizan para un tamaño de fichero de 380 Kb y una p de 10, 15, 20 y 25 dígitos. Se realizan 5 ejecuciones para cada valor de p para intentar paliar la aleatoriedad del sistema.

En la siguiente tabla se muestra el tiempo necesario para calcular la curva inicial sobre $\mathbb{F}_p$ y los parámetros $\{R_{priv}, E_{pub}, F, \}$, para distintos tamaños de p.

Dígitos	E_{init} (sg)	Parámetros (sg)
10	480,7	481,67
15	3036,8	3037,26
20	8845,61	8846,96
25	18927,28	18928,081

Tabla 5.5: Tiempos en el cálculo de los parámetros comunes.

Hay que comentar que el tiempo de calcular los parámetros incluye el de la curva inicial, por lo que vemos que casi la totalidad del tiempo se usa en encontrar una curva elíptica inicial que cumpla con los requisitos. Se puede ver que cuanto más grande es p más tarda en calcular la curva. Esto es debido a que el cálculo del class number tiene un coste muy elevado, además a esto hay que añadirle la dificultad de encontrar una curva con discriminante primo y que cumpla con el simbolo de Kronecker para ciertos valores de ℓ. Los valores de L elegidos son $\{3,5,7,11,13,17\}$.

Dígitos	Cifrado (sg)	Descifrado (sg)
10	3,843	8,98
15	2,65	21,53
20	3,533	43,875
25	3,05	46,48

Tabla 5.6: Tiempos del cifrado y descifrado.

Como vemos el cifrado es bastante rápido y cuanto mayor es p más rápido es. Esto se debe a que para un p grande se cifran más carácteres de vez, lo que implica menos bloques para cifrar. En cuanto al descifrado, es mucho más lento que el cifrado. Cuando obtengo una línea del mensaje cifrado la paso de string a carácter mediante la función StringToInteger() de Magma. Este entero lo tendre que convertir a un elemento del cuerpo finito $\mathbb{F}_p$ (s:=GF(p)!s1;), esta es la operación que ralentiza tanto el descifrado, requiriendo la mayor parte del tiempo utilizado en descifrar la información.

Por último también se ha comparado el tamaño del fichero con el mensaje original y el del mensaje cifrado. Como se puede ver en la siguiente tabla el mensaje cifrado ocupa mucho más espacio que el mensaje en claro, siendo la tasa de expansión unas 3 veces. Esto se debe a que para el mensaje cifrado guardamos A_{add}, B_{add} y por cada trozo cifrado se guarda s. Tambien influye el tamaño de p, ya que cuanto más grande sea p más carácteres se podrán cifrar a la vez y por lo tanto habra menos bloques para cifrar.

Dígitos	Mensaje Original	Mensaje Cifrado
10	380 Kb	1,12 Mb
15	380 Kb	1047 Kb
20	380 Kb	994 Kb
25	380 Kb	982 Kb

Tabla 5.7: Tamaño del mensaje original y el cifrado.

5.3. Conclusiones y futuras líneas de trabajo

El objetivo de este proyecto era implementar un criptosistema de clave pública basado en estrellas de isogenias de forma eficiente utilizando el software matemático MAGMA. La implementación incluye el cálculo de curvas isogenas y a partir de sus coeficientes, la obtención de su kernel. Esta ha sido una de las partes a la que más tiempo se ha dedicado ya que es un cálculo bastante complejo. Otra de las partes importantes de este trabajo es el cálculo de un conjunto de curvas iniciales que nos darán lugar a los parámetros comunes y las claves. El problema en esta parte radica en el hecho de que el cálculo de una curva inicial sobre $\mathbb{F}_p$ junto con el conjunto L de grados de isogenias tiene un coste muy elevado cuando el tamaño de p y el cardinal de L son grandes. A pesar de todo esto, el resultado obtenido es muy positivo, ya que se han cumplido los objetivos iniciales y se ha conseguido realizar pruebas para valores aceptables de p.

Como futura línea de trabajo se propone mejorar la eficiencia en la obtención de una curva inicial con las propiedades requeridas utilizando técnicas de paralelización, para reducir el tiempo y de esta manera poder utilizar un tamaño de p mayor, lo que nos proporcionara una mayor seguridad.

Otra posible mejora sería utilizar un volcán de isogenias de altura mayor que 0, en vez de una estrella. De esta forma podríamos conseguir eliminar la restricción de que el class number sea primo.

Capítulo 6

Apéndice

A continuación se muestra toda la implementación.

setup.m

```
load "funciones.m";

CalculaP:= function()
           paux:=Random(10^20,10^21 - 1);
           p:=NextPrime(paux);
           return p;
end function;

CurvaInicial:= function(p)
               repeat
                      a:=Random(p);
                      b:=Random(p);
               until IsEllipticCurve([GF(p)|a,b]);
               e:=EllipticCurve([GF(p)|a,b]);
               return e;
end function;

Discriminante:= function(e,p)
                t:=TraceOfFrobenius(e);
                D:=t^2-4*p;
                return D;
end function;

GetClassNumber:= function(D)
```

```
                k:=QuadraticField(D);
                cn:=ClassNumber(k);
                return cn;
end function;

SimKronecker:=function(D,l)
             for i in [1..#l] do
                  if (Kronecker Symbol(D,l[i]) ne 1) then
                              return false;
                  end if;
             end for;
             return true;
end function;

ValoresPropios:= function(e,l,p)
                vp:=[0];
                for i in [1..#l] do
                         pr:=PolynomialRing(GF(l[i]));
                         t:=TraceOfFrobenius(e);
                         pol:=x^2-t*x+p;
                         sol:=Roots(pol);
                         vp[i]:=sol[1][1];
                end for;
                return vp;
end function;

GuardarClaves:= procedure(r,apub,bpub)
               f1:=Open("claves/ClavePublica.txt","w");
               f2:=Open("claves/ClavePrivada.txt","w");
               Puts(f1,Sprint(apub));
               Puts(f1,Sprint(bpub));
               for i in [1..#r] do
                     Put(f2,Sprint(r[i]));
                     Put(f2,);s
               end for;
               Puts(f2,);
               delete f1;
               delete f2;
end procedure;

GuardarParams:= procedure(p,e,d,l,dir)
               f:=Open("setup/parametros.txt","w");
               a:=aInvariants(e);
```

```
                a1:=a[4];
                a2:=a[5];
                Puts(f,Sprint(p));
                Puts(f,Sprint(a1));
                Puts(f,Sprint(a2));
                Puts(f,Sprint(d));
                for i in [1..#l] do
                      Put(f,Sprint(l[i]));
                      Put(f," ");
                end for;
                Puts(f," ");
                for i in [1..#l] do
                      Put(f,Sprint(dir[i]));
                      Put(f," ");
                end for;
                Puts(f," ");
                delete f;
end procedure;

GuardarCurva:= procedure(e,p,s)
               f:=Open("curvas/"*s,"w");
               a:=aInvariants(e);
               a1:=a[4];
               a2:=a[5];
               Puts(f,Sprint(p));
               Puts(f,Sprint(a1));
               Puts(f,Sprint(a2));
               delete f;
end procedure;

CalRuta:=function(a,b)
               repeat
                      r:=[0];
                      q:=0;
                      for i in[1..a] do
                            r[i]:=Random(b);
                            if r[i] eq 0 then
                                    q:=q+1;
                            end if;
                      end for;
               until q ne a;
               return r;
end function;
```

```
//Calcula la curva inicial, los parametros y las claves
Calcular:= procedure(s)
           l:=[3,5,7,11,13,17];
           repeat
                 repeat
                        p:=CalculaP();
                        e:=CurvaInicial(p);
                        D:=Discriminante(e,p);
                 until IsPrime(D) and SimKronecker(D,l);
                 cn:=GetClassNumber(D);
           until IsPrime(cn);
           GuardarCurva(e,p,s);
           dir:=ValoresPropios(e,l,p);
           r:=CalRuta(#l,10);
           apub,bpub:=EllipticCrv(r,e,l,p,dir);
           GuardarClaves(r,apub,bpub);
           GuardarParams(p,e,#l,l,dir);
end procedure;

//Obtiene una curva incial ya calculada
leer:= function(s)
       f:=Open("curvas/"*s,"r");
       p:=StringToInteger(Gets(f));
       a:=StringToInteger(Gets(f));
       b:=StringToInteger(Gets(f));
       w:=EllipticCurve([GF(p)|a,b]);
       delete f;
       return w,p;
end function;

//Calcula los parametros y las claves a partir de una curva inicial
CalcularParametros:=procedure(s)
       e,p:=leer(s);
       l:=[3,5,7,11,13,17];
       D:=Discriminante(e,p);
       cn:=GetClassNumber(D);
       dir:=ValoresPropios(e,l,p);
       r:=CalRuta(#l,10);
       apub,bpub:=EllipticCrv(r,e,l,p,dir);
       GuardarClaves(r,apub,bpub);
       GuardarParams(p,e,#l,l,dir);
end procedure;
```

cifrado.m

```
load "funciones.m";

ObtenerParams:= function()
                f1:=Open("claves/ClavePublica.txt","r");
                f2:=Open("setup/parametros.txt","r");
                apub:=StringToInteger(Gets(f1));
                bpub:=StringToInteger(Gets(f1));
                p:=StringToInteger(Gets(f2));
                a:=StringToInteger(Gets(f2));
                b:=StringToInteger(Gets(f2));
                d:=StringToInteger(Gets(f2));
                e:=EllipticCurve([GF(p)|a,b]);
                epub:=EllipticCurve([GF(p)|apub,bpub]);
                l:=StringToIntegerSequence(Gets(f2));
                dir:=StringToIntegerSequence(Gets(f2));
                delete f1;
                delete f2;
                return epub,p,e,l,dir;
end function;

//Pasa un texto a entero
dcae:= function(s)
       total:=0;
       for i in [0..#s-1] do
             c:=StringToCode(s[i+1]);
             pw:=256^i;
             mul:=c*pw;
             total:=total+mul;
       end for;
       return total;
end function;

guardarEadd:= procedure(aadd,badd,f1)
              Puts(f1,Sprint(aadd));
              Puts(f1,Sprint(badd));
end procedure;

guardarmen:= procedure(s,f1)
             Puts(f1,Sprint(s));
end procedure;
```

```
//Calcula el numero maximo de caracteres que se pueden cifrar
num_max_de_car:= function(p)
                 max:=((Log(2,255*p+1))/8)-1;
                 max:=Truncate(max);
                 return max;
end function;

CalRutaCi:= function(a,b)
            repeat
                   r:=[0];
                   q:=0;
                   for i in[1..a] do
                         r[i]:=Random(b);
                         if r[i] eq 0 then
                                 q:=q+1;
                         end if;
                   end for;
            until q ne a;
            return r;
end function;

Cifrar_cadenas :=procedure(s,f1,jenc)
                m:=dcae(s);
                s:=jenc*m;
                guardarmen(s,f1);
end procedure;

Cifrar:=procedure()
        epub,p,e,l,dir:=ObtenerParams();
        renc:=CalRutaCi(#l,10);
        aenc,benc:=EllipticCrv(renc,epub,l,p,dir);
        jenc:=jInvariant(EllipticCurve([GF(p)|aenc,benc]));
        aadd,badd:=EllipticCrv(renc,e,l,p,dir);
        num_max:=num_max_de_car(p);
        f:=Open("mensajes/mensaje.txt","r");
        f1:=Open("mensajes/men_cifrado.txt","w");
        guardarEadd(aadd,badd,f1);
        c1:=Getc(f);
        Rewind(f);
        while not IsEof(c1) do
              i:=0;
              s:=" ";
```

```
            while (not IsEof(s) and i lt num_max) do
                  c:=Getc(f);
                  if IsEof(c) then
                              break;
                  end if;
                  s:=s*c;
                  i:=i+1;
            end while;
            Cifrar_cadenas(s,f1,jenc);
            c1:=Getc(f);
            if not IsEof(c1) then
                   Ungetc(f,c1);
            end if;
      end while;
      delete f;
      delete f1;
      end procedure;
```

descifrado.m

```
load "funciones.m";

ObtenerParams:=function()
            f1:=Open("setup/parametros.txt","r");
            f2:=Open("claves/ClavePrivada.txt","r");
            p:=StringToInteger(Gets(f1));
            a:=StringToInteger(Gets(f1));
            b:=StringToInteger(Gets(f1));
            d:=StringToInteger(Gets(f1));
            e:=EllipticCurve([GF(p)|a,b]);
            l:=StringToIntegerSequence(Gets(f1));
            dir:=StringToIntegerSequence(Gets(f1));
            cn:=StringToInteger(Gets(f1));
            delete f1;
            delete f2;
            return p,e,l,rpriv,dir;
end function;


Obtener_Eadd:=function(f1)
           aadd:=Gets(f1);
           badd:=Gets(f1);
           return aadd,badd;
```

```
end function;

Obtener_MenCi:=function(f1)
             s:=Gets(f1);
             return s;
end function;

//Funcion que pasa un valor decimal a un string de caracteres
deac:=procedure(m,f)
             repeat
                   aux:=m mod 256;
                   m:=m div 256;
                   c:=CodeToString(aux);
                   Put(f,Sprint(c));
             until (m le 0);
end procedure;

Descifrar:=procedure()
           p,e,l,rpriv,dir:=ObtenerParams();
           f:=Open("mensajes/men_descifrado.txt","w");
           f1:=Open("mensajes/men_cifrado.txt","r");
           aadd,badd:=Obtener_Eadd(f1);
           aadd:=GF(p)!StringToInteger(aadd);
           badd:=GF(p)!StringToInteger(badd);
           eadd:=EllipticCurve([GF(p)|aadd,badd]);
           aenc,benc:=EllipticCrv(rpriv,eadd,l,p,dir);
           jenc:=jInvariant(EllipticCurve([GF(p)|aenc,benc]));
           s1:=Obtener_MenCi(f1);
           while not IsEof(s1) do
                 s:=GF(p)!StringToInteger(s1);
                 m:=s/jenc;
                 I:=IntegerRing();
                 m:=I!m;
                 deac(m,f);
                 s1:=Obtener_MenCi(f1);
           end while;
           delete f;
           delete f1;
end procedure;
```

funciones.m

```
//Cacula la isogenia (a_tilde,b_tilde) y p1
CodomainCanonical:=function(e,l,p,phic,ef)
                // e : Curva eliptica,
                // ell : grado de la isogenia,
                // phic : canonical modular polynomial
                // ef : raiz de phic que define la isagenia Eb.
                s_val:=12 div Gcd(12,l-1);
                v:=s_val*(l-1) div 12;
                j:=jInvariant(e);
                ef_tilde:=l^(s_val)/ef;
                DFpoly:=Derivative(phic,1);
                DF:=ef*Evaluate(DFpoly,[ef,j]);
                DJpoly:=Derivative(phic,2);
                DJ:=j*Evaluate(DJpoly,[ef,j]);
                DFFpoly:=ef^2*Derivative(DFpoly,1);
                DFF:=DF+Evaluate(DFFpoly,[ef,j]);
                DFJpoly:=ef*j*Derivative(DJpoly,1);
                DFJ:=Evaluate(DFJpoly,[ef,j]);
                DJJpoly:=j^2*Derivative(DJpoly,2);
                DJJ:=DJ+Evaluate(DJJpoly,[ef,j]);

                _,_,_,e4,e6:=Explode(Coefficients(e));
                E4:=-e4/3;
                E6:=-e6/2;
                E4_tilde:=(E4+((144*DJ^2*E6^2)/(s_val^2*DF^2*E4^2))-((48*DJ*E6^2)/(s_val*DF*E4^2))
                        -((288*DJ*DFJ*E6^2)/(s_val*DF^2*E4^2))+((144*DJJ*E6^2)/(s_val*DF*E4^2))
                        +((72*DJ*E4)/(s_val*DF))+((144*DJ^2*DFF*E6^2)/(s_val*DF^3*E4^2)))/l^2;
                delta:=(E4^3-E6^2)/1728;
                delta_tilde:=ef^(12 div s_val)*delta/(l^12);
                j_tilde:=E4_tilde^3/delta_tilde;
                E6_tilde:=(-E4_tilde)*((l^s_val /ef)*Evaluate(DFpoly,[(lŝ_val /ef),j_tilde])*DJ*E6)
                        /(l*j_tilde*Evaluate(DJpoly,[(l^s_val /ef),j_tilde])*DF*E4);
                new_a:=-3*l^4*E4_tilde;
                new_b:=-2*l^6*E6_tilde;
                Eb:=EllipticCurve([new_a,new_b]);
                p1:=(6*l*E6*DJ)/(s_val*E4*DF);
                return Eb, p1;
end function;


//Calcula el kernel de la isogenia
```

```
WeierstrassFunction:=function(e,prec)
          Q:=LaurentSeriesRing(BaseRing(e));
          z:=Q.1;
          _,_,_,a4,a6:=Explode(Coefficients(e));
          c:=[-a4/5,-a6/7];
          for k in [3..prec-1] do
                Append(~ c,3*(&+[c[h]*c[k-1-h]:h in [1..k-2]])/((k-2) * (2*k + 3)));
          end for;
          return z^(-2)+&+[c[k]*z^(2*k):k in [1..prec-1]]+O(z^(2*prec));
end function;

KernelPolynomial:=function(e,l,iso,p1)
               FF:=BaseRing(e);
               xtr:=2;
               deg:=(l-1) div 2;
               _,_,_,e4,e6:=Explode(Coefficients(e));
               _,_,_,e4_tilde,e6_tilde:=Explode(Coefficients(iso));
               c:=WeierstrassFunction(e,deg+xtr+1);
               c_tilde:=WeierstrassFunction(iso,deg+xtr+1);
               P:=PolynomialRing(FF);
               x:=P.1;
               psi2:=4*x^3+4*e4*x+4*e6;
               dpsi2:=6*x^2+2*e4;
               mulist:=[2*e4+6*x^2];
               for k in[2..deg+xtr] do
                       pp := Eltseq(mulist[k-1]);
                       np := pp[2] * dpsi2;
                       for ii in [3..#pp] do
                              i := ii - 1;
                              np +:= i*pp[i+1]*(dpsi2*x^(i-1) + (i-1)*psi2*x^(i-2));
                       end for;
                       Append(~mulist, np);
               end for;

               M := KMatrixSpace(FF, deg+xtr, deg+2+xtr)!0;
               for k in [1..deg+xtr] do
                     pp := Eltseq(mulist[k]);
                     for i in [1..#pp] do
                           M[k,i] := pp[i] * 2 / Factorial(2*k);
                     end for;
               end for;
               B := Vector([Coefficient(c_tilde,2*i)-Coefficient(c,2*i) : i in [1..deg+xtr]]);
               v_raw, V := Solution(Transpose(M), B);
```

```
                p0 := deg;
                v := v_raw + V.1 * (p0 - Eltseq(v_raw)[1]);
                v := v + V.2 * (p1 - Eltseq(v)[2]);
                ps := Eltseq(v);

                // Using Newton sums
                tlist := [ FF!0 : i in [1..deg+1+xtr] ];
                tlist[deg+1+xtr] := FF!1;
                for k in [1..deg+xtr] do
                      tmp := FF!0;
                      for i in [1..k-1] do
                            tmp := tmp + tlist[deg+xtr-i+1] * ps[k-i+1];
                      end for;
                      tmp := -tmp - ps[k+1];
                      tlist[deg+xtr-k+1] := tmp / k;
                end for;
                g<x>:= P![tlist[i] : i in [xtr+1..deg+xtr+1]];
                // Verify correctness.
                success := [ tlist[i] : i in [1..xtr] ] eq [ FF!0 : i in [1..xtr] ];
                return g, success;
end function;

//Calcula que direccion debe ser la isogenia
Direccion:=function(l,p,e,psi,dir)
         Z:=IntegerRing();
         pi:=Z!dir;
         P<x>:=PolynomialRing(GF(p));
         Q:=quo<P|psi>;
         xp:=Modexp(x,p,psi)-x;
         Q:=Parent(xp);
         psi_2 := Q!DivisionPolynomial(e,2);

         //Calculamos X
         _,tmp:=DivisionPolynomial(e,pi-1,psi);
         pdN:=Q!tmp;
         _,tmp:=DivisionPolynomial(e,pi,psi);
         pdC:=Q!tmp;
         _,tmp:=DivisionPolynomial(e,pi+1,psi);
         pdP:=Q!tmp;
         prod1:=xp*pdC*pdC;
         prod2:=pdN*pdP;
         if IsEven(pi) then
```

```
                ff:=prod1*psi_2+prod2;
        else
            ff:=prod1+psi_2*prod2;
        end if;
        _,restox:=Quotrem(ff,psi);

        //Calculamos la Y
        if pi eq1 then
                fkm2:=-1;
        else
             _, fkm2 := DivisionPolynomial(e, pi-2, psi);
        end if;
        _, fkm1 := DivisionPolynomial(e, pi-1, psi);
        _, fk := DivisionPolynomial(e, pi, psi);
        _, fkp1 := DivisionPolynomial(e, pi+1, psi);
        _, fkp2 := DivisionPolynomial(e, pi+2, psi);
        prod3 := fkp2*fkm1*fkm1 - fkm2*fkp1*fkp1;
        if IsOdd(pi) then
                pow2 := Modexp(psi_2, ((p-1) div 2), psi);
        else
            pow2 := Modexp(psi_2, ((p+3) div 2), psi);
        end if;
        pol:=pow2*fk^3-prod3;
        restoy:=pol mod psi;
        return restox,restoy;
end function;

CalIsogenia:=function(e,l,p,dir)
          PR<x,y>:=PolynomialRing(GF(p),2);
          CMP<x,y>:=CanonicalModularPolynomial(l);
          phic:=PR!CMP;
          j:=jInvariant(e);
          CMP_x:=Evaluate(phic,2,j);
          f:=Factorization(CMP_x);
          g:=[];
          //Cacula las dos isogenias y el kernel de cada una
          for i in [1..2] do
                raiz:=f[i][1];
                raiz:=Coefficients(raiz,1);
                raiz:=GF(p)![-raiz[1]];
                ee,p1:=CodomainCanonical(e,l,p,phic,raiz);
                g[i],suc:=KernelPolynomial(e,l,ee,p1);
          end for;
```

```
            //Elegimos con que kernel nos quedamos
            for i in [1..2] do
                  x1,y1:=Direccion(l,p,e,g[i],dir);
                  if x1 eq 0 and y1 eq 0 then
                        psi:=g[i];
                        break;
                  end if;
            end for;
            //Por ultimo se obtiene la isogenia a partir del nucleo.
            i:= IsogenyFromKernel(e, psi);
            return i;
end function;

//Obtiene una curva eliptica a partir de una ruta y la estrella de isogenias.
EllipticCrv:=function(r,e,l,p,dir)
                  j:=e;
                  for z in [1..#l] do
                        for i in [1..r[z]] do
                              j:=CalIsogenia(j,l[z],p,dir[z]);
                              //Si llegamos al final del ciclo, seguimos desde el principio
                              if(IsIsomorphic(j,e)) then
                                    j:=e;
                                    mov:=r[z]-i;
                                    for k in [1..mov] do
                                          j:=CalIsogenia(j,l[z],p,dir[z]);
                                    end for;
                                    break;
                              end if;
                        end for;
                  end for;
                  ainv:=aInvariants(j);
                  a:=ainv[4];
                  b:=ainv[5];
                  return a,b;
            end function;
```

Bibliografía

[1] Blake I.,Seroussi G., Smart N., *Elliptic Curves in Cryptography*. Cambridge university press, 1999.

[2] Couveigness J.M., Dewaghe L., Morain F. *Isogeny cicles and the Schoof-Elkies-Atkin algorithm.* Ecole polytechnique, France, 1996.

[3] Diffie, W. and Hellman, M.E., *New directions in Cryptography. IEEE Transactions on Information Theory,* vol. IT-22, 1976, 644-654.

[4] Elkies N., *Elliptic and modular curves over finite fields and related computational issues.* Computational Perspectives on Number theory: Proceedings of a Conference in honor of A.O.L. Atkin (D.A. Buell and J.T. Teitelbaum, eds.; AMS/International Press, 1998). pp 21-76.

[5] Foutquet M., *Anneau d'endomorphismes et cardinalité des courbes elliptiques: Aspects algorithmiques*. PhD Thesis, Ecole polytechnique, France, 2001.

[6] Galbraith S., *Constructing isogenies between elliptic curves over finite fields*. London Math. Soc., Journal of Computational Mathematics, Vol 2, p. 118-138 (1999). http://www.lms.ac.uk/jcm/

[7] Koblitz, N., *Elliptic Curves Cruptosystems,* Mathematics of Computation, 48 (1987), 203-209.

[8] Lang S., *Elliptic functions*. Addison-Wesley, 1973.

[9] Lecier R., Morain F., *Algorithms For Computing Isogenies Between Elliptic Curves.* 1996.

[10] MAGMA Computational Algebra System Home Page: http://magma.maths.usyd.edu.au/

[11] Menezes A. J., *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[12] Miller V., *Use of elliptic curves in cryptography*, CRYPTO 85, 1985.

[13] Milne S.J., *Modular functions and modular forms.* http://www.jmilne.org/math/CourseNotes/math678.pdf.

[14] Muller V., *Ein Algorithmus zur Bestimmung der Punktanzahl elliptisher Kurven uber endlichen Korpen der Charakteristik groser drei.* Saarbrucken, 1995. http://www.informatik.tu-darmstadt.de/TI/Forschung/ECC.

[15] Rio A., *ℓ-division in elliptic curves over finite fields*. 2006.

[16] Rivest, R., Shamir, A. and Adleman, L., *A method for obtaining digital signatures and PKC. Comunications of the ACM.* Vol. 21 (2), 1978, 120-128.

[17] Rostovtsev A., Stolbunov A., *Public-Key Cryptosystem based on Isogenies*. Saint-Petersburg State Polytechnical University, Russia.

[18] Shanks D., *Class number, a theory of factorisation, and genera*. Proceedings of the symposium on pure mathematics, vol 20, AMS, 1971, pp. 415-440.

[19] Shor P.W., *Polynomial Time Algorithms For Prime Factorization And Discrete Logarithms On A Quantum Computer,* SIAM J. Sci. Statist. Comput. 26 (1997) 1484.